Disambiguation Data: Extracting Information from Anonymized Sources

Stephan Dreiseitl*, PhD, Staal Vinterbo†‡, PhD, Lucila Ohno-Machado†‡, MD, MHA, PhD

*** Polytechnic University of Upper Austria at Hagenberg**
Dept. of Software Engineering for Medicine
A-4232 Hagenberg, Austria

† Decision Systems Group, Brigham and Women's Hospital
‡ Harvard Medical School
Boston, MA, 02115

email: Stephan.Dreiseitl@fhs-hagenberg.ac.at

Abstract

Privacy protection is an important consideration when releasing medical databases to the research community. We show that while recent advances in anonymization algorithms provide increased levels of protection, it is still possible to calculate approximations to the original data set. In some cases, one can even uniquely reconstruct entries in a table before anonymization.

In this paper, we demonstrate how knowledge of an anonymization algorithm based on ambiguating data cell entries can be used to undo the anonymization process. We investigate the effect of this algorithm and its reversal on data sets of varying sizes and distributions. It is shown that by using a computationally complex disambiguation process, information on individuals can be extracted from an anonymized data set.

1 Introduction

With evidence-based medicine becoming more and more important, the widespread dissemination of medical information is becoming more and more important. The need for dissemination pertains not only to the results and findings obtained from data, but to the data itself. When only few patient cases are documented in a study, it becomes important to pool several studies to obtain a sufficiently large data set. Such data sets overcome the limitations imposed by smaller data sets, making statistical methods more accurate.[1]

While freely granting access to medical data can only be beneficial to the public, the right of the individual to privacy protection must not be violated.[2–4] Therefore, data sets are "scrubbed" before released to the public; i.e., any unique identifiers in the data are removed. It has been shown, however, that this is not sufficient to completely anonymize the data.[5] Given other data tables that contain unique identifiers as well as some of the variables in the anonymized table, it is often possible to link both tables and thus reconstruct identifiers for the original table.

Over the last couple of years, there have been several approaches to remedy this situation by either generalization,[6,7] column suppression,[8] or encryption.[9] Some of the most promising procedures for ambiguating data tables cell suppression algorithms.[10–12] This approach strikes a balance between wanting to release as much information as possible (for research purposes) and restricting access as much as possible (for privacy protection). Cell suppression works by blanking certain fields in the data table in such a way that no entry (row) in the table is unique. This makes it impossible to uniquely identify an entry by linking to another data table, since in an ambiguated table, at least two rows will match any linking operation.

In this paper, we will focus only on privacy protection by cell suppression and demonstrate that data tables ambiguated by this method are not immune to disambiguation efforts. For this, we will provide a deeper understanding of the cell suppression algorithm in Section 2, followed by a presentation of how the effects of ambiguation can be reversed in Section 3. Section 4 presents experiments on data sets of varying sizes and distributions, showing that ambiguation by cell suppression can indeed be undone. A discussion in Section 5 concludes the paper.

1067—5027/01/$5.00

2 Minimal Cell Suppression

The need for additional levels of privacy protection (after eliminating unique identifiers from a table) can be illustrated in a simple example. Suppose that a database containing the following demographic data and medical condition is released to the research community. Further assume that there are no other rows with the same combination of values in the table.

age	gender	ZIP code	med. condition
20-30	m	02215	0
20-30	m	02215	1
50-60	m	02115	1
20-30	f	02115	0

Now if someone has access to a database containing date of birth, gender and ZIP code information, and knows that a specific person has been tested for the medical study reported in the table, it is sometimes possible to uniquely identify this person and his/her medical condition. A real-world example of this scenario is published in the literature.[5] In our example, unique identification is possible in rows 3 and 4 of the table above, but not in rows 1 and 2, since they contain the same demographic data. In the following, we will say that two rows are identical if they agree on their demographic data. Similarly, a row is unique if there is no other row with the same entries in the demographic data columns.

This attack on privacy protection is possible because there are unique rows in the table. The simplest approach to diverting it would thus be to eliminate all unique rows from the table before releasing it. However, such an approach is impractical, as the number of unique rows grows with the number of non-redundant attributes (columns) in the table. This is shown graphically in Figure 1. In databases with more than a few columns, deleting unique rows would then be equivalent to discarding large portions of the database.

A more viable approach is to ambiguate certain table entries in such a way that there are no more unique rows in the table. To ambiguate an entry means to replace this entry with a special symbol $\perp$ that is, by definition, indiscernible from all other entries. In the above data table, one could replace entries with these "wild card" symbols in the following way:

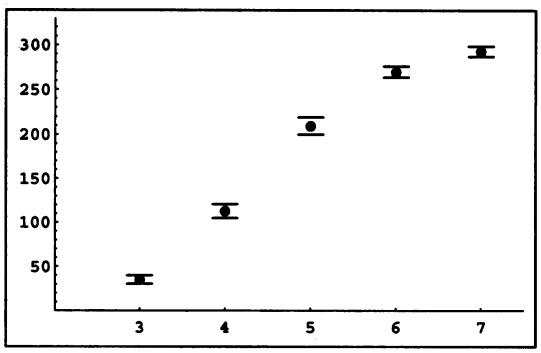

Figure 1: For tables with constant number of rows (here: 300), the number of unique rows grows as new columns are added. Shown here are the average number of unique rows (and their standard deviations over 10 iterations) in tables of normally distributed ordinal data with 3 to 7 columns.

age	gender	ZIP code	med. condition
20-30	m	02215	0
20-30	m	02215	1
50-60	$\perp$	02115	1
$\perp$	f	02115	0

This process of ambiguation data is called *minimal cell suppression*.[11,12] Since the $\perp$ symbols match all other entries, the last two rows are thus indiscernable and no longer unique. Note that the minimum number of $\perp$ symbols to ambiguate this table is two; these two symbols could have also replaced the first entry in row 3 (50-60) and the second entry in row 4 (f).

The drawback of minimal cell suppression is that it achieves privacy protection by introducing missing values into the data. It is, however, important to note that there are far fewer "holes" in the ambiguated table than unique rows before ambiguation. This is due to the fact that a row with a $\perp$ symbol can match more than one unique row. The number of $\perp$ symbols in a data table grows with both the number of rows and the number of columns in the table. However, when regarding the fraction of ambiguated entries in the data table, it can be seen that the percentage of $\perp$ symbols grows with the number of columns, but decreases with the number of rows. This is because each new row is less likely to be unique, whereas each new uncorrelated column increases the number of rows that have to be ambiguated. This fact is illustrated for tables with a fixed number of rows in Figure 2 and for a fixed number of columns in Figure 3.

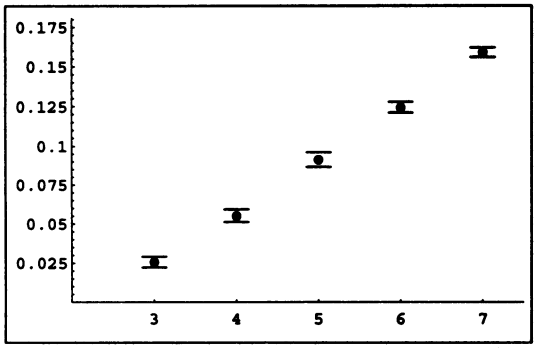

Figure 2: Number of ambiguated entries for data tables with fixed number of rows (here: 300) as fraction of total number of entries, for varying number of columns. Shown here are averages (and their standard deviations over 10 iterations) in tables of normally distributed ordinal data with 3 to 7 columns.

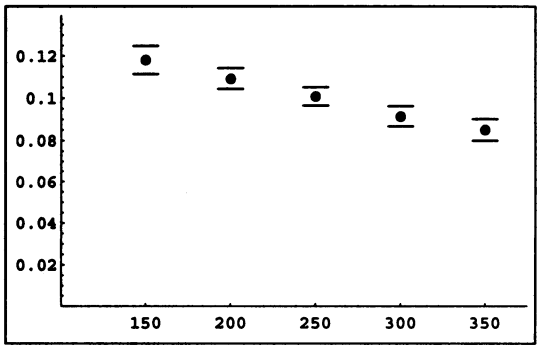

Figure 3: Number of ambiguated entries for data tables with fixed number of columns (here: 5) as fraction of total number of entries, for varying number of rows. Shown here are averages (and their standard deviations over 10 iterations) in tables of normally distributed ordinal data with 150 to 350 rows.

For increased levels of protection, it is possible to extend the argument above to require that every row is indiscernible from two or more other rows, and not just from one other row. This extension requires additional $\perp$ symbols in the table.

The problem for finding the minimum number of $\perp$ symbols and their optimal placement to ambiguate a given data table is NP-hard and can therefore not be solved optimally in reasonable time for all but small examples. It is, however, possible to use heuristics to find nearly optimal solutions.[11,12]

In the next section, we will show that while minimal cell suppression algorithms can be used to ambiguate data tables before releasing them to the research community, they cannot guarantee complete privacy protection.

3 Disambiguating Anonymized Data

Consider the following simple ambiguated data table:

age	gender	ZIP code	med. condition
20-30	m	02215	0
20-30	$\perp$	02215	1

Now it is immediately obvious that the missing value $\perp$ could only have had the value f in the original (un-ambiguated) table: Since the only two possible values for the attribute "gender" are m and f, and replacing $\perp$ by m would have made the two rows indiscernible in the original table (and therefore no ambiguation would have been necessary), we can conclude that the original entry had been f.

The approach used to disambiguate the above example can be formalized and applied to larger data tables as well. The key to this attack at privacy-protected data is knowledge of the protection algorithm: The above argument relies on knowing that the entries in the original table had been ambiguated by minimal cell suppression. Given this knowledge, one can construct the following algorithm for disambiguating a given data table:

1. determine the possible values that each $\perp$ symbol can represent, i.e., the range of attribute values in all columns that contain $\perp$ symbols,
2. generate all possible instantiations of the ambiguated table by replacing all $\perp$ symbols by the values they can represent,
3. eliminate all table instantiations that could not have led to the given ambiguated table, first by
   (a) checking whether a row containing a $\perp$ symbol is now unique in the instantiated table, and if this holds true for all rows, then by

(b) checking whether the ambiguation algorithm, when given the instantiated table as input, produces the given ambiguated table as output.

Note that step 3(b) is much more general than step 3(a), so that step 3(a) is not really necessary. The computational complexity of 3(a), however, is only a fraction of that of 3(b), and it therefore makes sense to check this simple condition first.

One can easily see that disambiguation is much more computationally demanding than ambiguation by noting that the number of iterations of the algorithm above grows exponentially with the number of $\perp$ symbols in the ambiguated table. This is due to the fact that each $\perp$ symbol has to be instantiated with all possible values independently of all other instantiations. An ambiguated two-column data set with attribute values $\{-5, \ldots, 5\}$ in both columns and four $\perp$ symbols must therefore be instantiated $4^{11} = 4,194,304$ times. Given that each iteration could potentially call the ambiguation algorithm in step 3(b), it is obvious that disambiguation is computationally demanding.

Given an ambiguated data table, the result of disambiguation is a set of all possible instantiated tables that, when used as input for the ambiguation algorithm, produce the given ambiguated table as output. If this set contains only one instantiation, then there is a unique value for all $\perp$ symbols. Otherwise, there are rows that were instantiated in more than one way. For these rows, one can only give distributions of their (instantiated) values. The conclusions to be drawn from such cases are statistical, such as determining the most likely value for a $\perp$ symbol.

4 Experimental Results

Since the ambiguation algorithm uses a heuristic, it is hard to calculate which of the two situations above (one or many solutions to the disambiguation problem) is more likely to be encountered in practice. We therefore investigated the performance of the disambiguation algorithm on two different data sets to see how data set size and distribution influence the results. Note that due to the computational complexity of the disambiguation algorithm, the data sets analyzed in this paper can only be seen as toy examples that serve to validate the approach presented here.

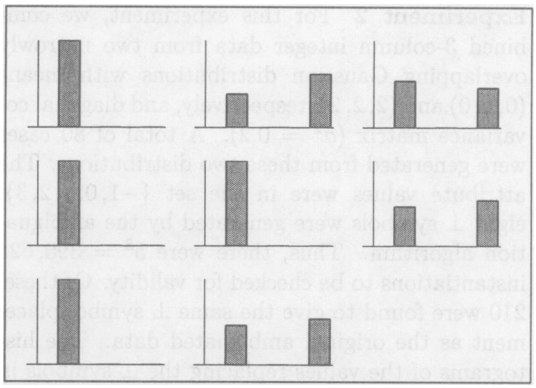

Figure 4: Histograms of values instantiated for each of the eight $\perp$ symbols in the data set of experiment 1.

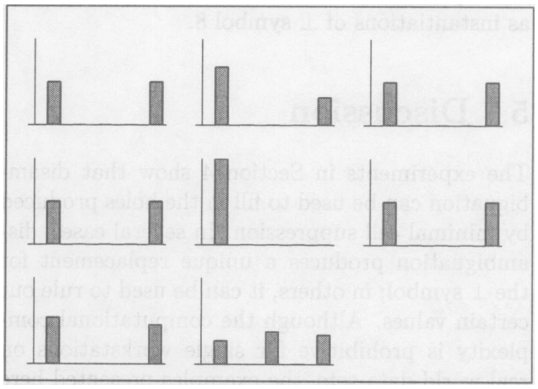

Figure 5: Histograms of values instantiated for each of the eight $\perp$ symbols in the data set of experiment 2.

Experiment 1 The 200-row, 5-column data in this experiment was drawn from a zero-mean normal distribution with diagonal covariance matrix with $\sigma^2 = 0.08$. To produce ordinal data, all real-valued random numbers were rounded to the closest integer. This meant that attribute values were limited to the set $\{-1, 0, 1\}$; eight $\perp$ symbols were required to ambiguate the data set. Disambiguation therefore only had to check $3^8 = 6,561$ instantiations. Of these, 14 produced the original table when ambiguated. The histograms of the values replacing the $\perp$ symbols in these instantiations are shown in Figure 4. One can see that for the $\perp$ symbols 1,3,5 and 7, the only possible instantiation is -1. For the other symbols, values -1 and 1 are almost equally likely.

Experiment 2 For this experiment, we combined 3-column integer data from two narrowly overlapping Gaussian distributions with means $(0,0,0)$ and $(2,2,2)$, respectively, and diagonal covariance matrix ($\sigma^2 = 0.2$). A total of 80 cases were generated from these two distributions. The attribute values were in the set $\{-1,0,1,2,3\}$; eight $\perp$ symbols were generated by the ambiguation algorithm. Thus, there were $5^8 = 390,625$ instantiations to be checked for validity. Of these, 210 were found to give the same $\perp$ symbol placement as the original ambiguated data. The histograms of the values replacing the $\perp$ symbols in these instantiations are shown in Figure 5. Except for $\perp$ symbol 5, where -2 is the only possible instantiation, all other $\perp$ symbols can be instantiated in at least two ways. One can, however, exclude values 0, 1 and 2 as possible instantiations of $\perp$ symbols 1,2,3,4,6 and 7, and values 0 and 2 as instantiations of $\perp$ symbol 8.

5 Discussion

The experiments in Section 4 show that disambiguation can be used to fill in the holes produced by minimal cell suppression. In several cases, disambiguation produces a unique replacement for the $\perp$ symbol; in others, it can be used to rule out certain values. Although the computational complexity is prohibitive for single workstations on real-world data sets, the examples presented here show that minimal cell suppression is not immune to a brute-force disambiguation effort, given the necessary computational resources. Higher levels of security are possible by combining cell suppression and other privacy protection algorithms. It will be the topic of further research to determine how much more resistant such combinations are to disambiguation efforts.

More detailed analysis of the disambiguation results can be obtained by noting that some of the instantiated rows are more likely than others, given that the data was produced by a specific distribution. Although this distribution is not known in practice, there are numerous density estimation or model fitting techniques that can be used to infer this distribution from the non-$\perp$ rows in the data table. It is then possible to rank instantiations based on the likelihood that they were produced by the inferred distribution. More research and experiments will be required to determine how this approach can be used to narrow down the solutions of the disambiguation algorithm.

References

[1] Normand S. Meta-analysis: formulating, evaluating, combining, and reporting. Stat Med 1999;18:321–359.

[2] Lako C. Privacy protection and population-based health research. Soc Sci Med 1986; 23:293–295.

[3] Buckovich S, Rippen H, Rozen M. Driving toward guiding principles: a goal for privacy, confidentiality, and security of health information. J Am Med Inform Assoc 1999;6:122–133.

[4] Korn D. Medical information privacy and the conduct of biomedical research. Acad Med 2000;75:963–968.

[5] Sweeney L. Weaving technology and policy together to maintain confidentiality. J Law Med Ethics 1997;25:98–110.

[6] Sweeney L. Replacing personally-identifying information in medical records, the scrub system. Proc AMIA Annu Fall Symp 1996;333–337.

[7] Sweeney L. Guaranteeing anonymity when sharing medical data, the datafly system. Proc AMIA Annu Fall Symp 1997;51–55.

[8] Su TA, Ozsoyoglu G. Controlling fd and mvd inferences in multilevel relational database systems. IEEE Transactions on Knowledge and Data Engineering 1991;3:474–485.

[9] Gulcher J, Kristjansson K, Gudbjartsson H, Stefansson K. Protection of privacy by third-party encryption in genetic research in Iceland. Eur J Hum Genet 2000;8:739–742.

[10] Fischetti M, Salazar J. Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. Mathematical Programming 1999; 84:283–312.

[11] Øhrn A, Ohno-Machado L. Using Boolean reasoning to anonymize databases. Artif Intell Med 1999;15:235–254.

[12] Vinterbo S, Ohno-Machado L. Table ambiguation by minimal cell suppression: problem characterization and effect on functional dependecies. Technical report, Decision Systems Group, Brigham and Women's Hospital, Harvard Medical School, Boston, 2001.